

# Balanced Operators

*Balanced problem from Eric Roberts*

In the syntax of most programming languages, there are characters that occur only in nested pairs, which are called *bracketing operators*. C++, for example, has these bracketing operators:

```
( . . . )  
[ . . . ]  
{ . . . }
```

The compiler validates that all bracketing operators are properly nested and each opener is matched to its closer. If you have ever failed to follow protocol, you have encountered the compiler's ire expressed in a swath of highlighted red text. For this assignment task, you will write a recursive function to confirm that the bracketing operators are balanced.

To process an input, first extract the sequence of parentheses, brackets, and braces, ignoring all other characters:

```
int main() { int x = 2 * (vec[2] + 3); x = (1 + random()); }
```

The above input filters to:

```
() {[ ]} ( ( ) )
```

A careful matching of opener to closer shows that this sequence of operators is correctly balanced.

The following strings, however, are unbalanced for the reasons stated below:

```
( ( [ a ] )   The line is missing a closing parenthesis.  
3 ) (         The closing parenthesis comes before the opening parenthesis.  
{ ( x } y )   The parentheses and braces are improperly nested.
```

The function **isBalanced** takes an input string **str** and returns **true** if the bracketing operators in **str** are balanced. Our provided code for **isBalanced** first calls **operatorsFrom** to extract the sequence of bracketing operators which is then passed to **operatorsAreMatched** to determine if the sequence of operators is balanced.

```
bool isBalanced(string str) {  
    string ops = operatorsFrom(str);  
    return operatorsAreMatched(ops);  
}
```

Your job is to implement the two recursive functions:

```
string operatorsFrom(string str);  
  
bool operatorsAreMatched(string ops);
```

The **operatorsFrom** function returns a string consisting of only the bracketing characters from **str**. You implemented various string cleaning operations in previous assignments, but in those situations you processed the string using an iterative loop. This time you are to take a **recursive** approach. Recursively processing a string takes this general form:

- Process the first character of the string and determine how/whether it contributes to the returned output/result.
- Make a recursive call to process the rest of the string and save that result.
- Combine the results (from first character and recursive rest) to form the result to return.

Writing this function is a good warmup for practicing recursive decomposition. **Be sure to thoroughly test your function before moving on. You should add at least 2-3 tests of your own to validate the behavior of your operatorsFrom function.**

Next up is the `operatorsAreMatched` function, which is also to be implemented **recursively**. Consider the following recursive insight about how to approach this task. A string consisting of only bracketing characters is balanced if and only if one of the following conditions holds:

- The string is empty.
- The string contains "`()`", "`[]`", or "`{}`" as a substring and the rest of the string is balanced after removing that substring.

For example, the operators "`[(){}]`" are shown to be balanced by the following chain of calls:

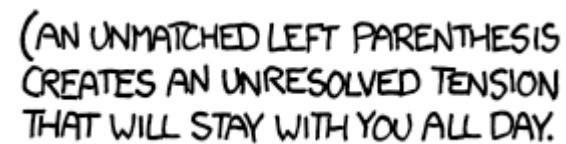
```
operatorsAreMatched("[(){}]") -> "()" exists, remove it and check whether rest is matched
operatorsAreMatched("[{}]") -> "{}" exists, remove it and check whether rest is matched
operatorsAreMatched("[]") -> "[]" exists, remove it and check whether rest is matched
operatorsAreMatched("") -> true
```

**Thoroughly test this function by add at least 3-4 tests of your own to validate the behavior of your operatorsAreMatched function.**

**Q7.** Compare your recursive solution to the iterative approach used for the Check Balance problem in the [Section 2](#). Which version do you find easier to read and understand? In which version did you find it easier to confirm the correct behavior?

## Notes

- Both of your functions must operate **recursively** and should not use a loop over the input string. You also should not use any auxiliary data structures (no Stacks, Queues, Vectors, etc.) to store or build up intermediate results.
- We have provided a few tests to get you started. As mentioned above, you will need to add student tests of your own for complete coverage. Given thorough independent testing of the `operatorsFrom` and `operatorsAreMatched` functions, you won't likely need much additional test coverage specific to `isBalanced`.
- The `operatorsAreMatched` function requires that the input string contain only bracketing operators. When given an input string that erroneously includes non-bracketing characters, `operatorsAreMatched` will return false. This is expected, as the input doesn't meet the function's precondition.



Courtesy of [xkcd](#).